**文本格式化工具AWK**

# AWK：

## 概述：

grep，sed，awk 是 shell 编程中经常用到的文本处理工具（shell编程三剑客）。 awk 命令名称来源于它的三个开发者 Aho、 Weinberger 和 Kernighan 名字首字母的缩写，它是一种编程语言，主要用来处理结构化的数据和生成格式化的报告。awk可以在非交互的情况下完成相当复杂的文本处理操作。

awk 提供了极其强大的功能：它可以完成 grep 和 sed 所能完成的所有工作，同时，还可以进行样式装入、流程控制、数学运算符、进程控制语句甚至于使用内置的变量和函数。

## 格式：

* awk 选项 ‘模式或条件{编辑指令}’ 输入文件
* awk -f 脚本文件 输入文件

## awk工作原理：

在 Linux 系统中/etc/passwd是一个非常典型的格式化文件，各字段（列）间使用“：”作为分隔符隔开， Linux 系统中的大部分日志文件也是格式化文件，从这些文件中提取相关信息是运维的日常工作内容之一。

**[root@localhost ~]# cat /etc/passwd**

**root:x:0:0:root:/root:/bin/bash**

**bin:x:1:1:bin:/bin:/sbin/nologin**

**daemon:x:2:2:daemon:/sbin:/sbin/nologin**

**adm:x:3:4:adm:/var/adm:/sbin/nologin**

**lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin**

**sync:x:5:0:sync:/sbin:/bin/sync**

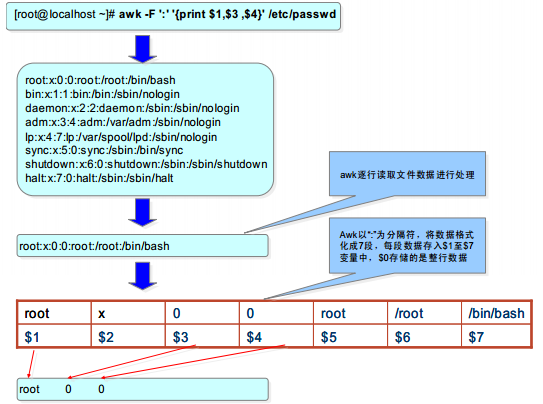
**… … …**

若需要查找出/etc/passwd 的用户名、用户 ID、组 ID等列，使用下面的 awk 命令就可完成：

**[root@localhost ~]# awk -F ':' '{print $1,$3,$4}' /etc/passwd**

awk 从输入文件或者标准输入中读入信息，与 sed 一样信息的读入也是逐行读取的。不同的是 awk 将文本文件中的一行视为一个记录，而将一行中的某一部分（列）作为记录中的一个字段（域）。为了操作这些不同的字段， awk 借用 shell 中类似于位置变量的方法，用$1， $2， $3…这样的顺序地表示行（记录）中的不同字段。另外awk 用$0 表示整个行（记录）。不同的字段之间是用称作分隔符的字符分隔开的。awk默认的分隔符是空格和制表符。 awk 允许在命令行中用[-F 分隔符]的形式来指定分隔符。

在上述示例中， awk 命令对/etc/passwd 的处理过程如图所示。



## 编辑指令的分隔：

* 编辑指令若包含多条命令语句，则以分号隔开。

[root@localhost ~]# free | awk '/^Mem:/{num=int($3/$2\*100); print "内存使用百分比："num"%"}'

内存使用百分比：34%

* 如果有多条编辑指令，则使用以分号或者空格分隔的多个{ }区域。

## 命令区块构成：

* BEGIN { 编辑指令 } //开始处理第一行文本之前的操作
* { 编辑指令 } //针对每一行文本的处理操作
* END { 编辑指令 } //处理完最后一行文本之后的操作

## awk的执行流程：

1. 首先执行 BEGIN { } 区块中的初始化操作；
2. 然后从指定的数据文件中循环读取一个数据行（自动更新NF、NR、$0、$1…… 等内建变量的值），并执行'模式或条件{ 编辑指令 }'；
3. 最后执行 END { } 区块中的后续处理操作

awk在使用的过程中，可以使用逻辑操作符&&，表示“逻辑与”，||表示“逻辑或”，!表示“逻辑非”；还可以进行简单的数学运算，如+、-、\*、/、%、^分别表示加、减、乘、除、取余、乘方。

## Awk练习文本：

**[root@localhost ~]# cat grade.txt**

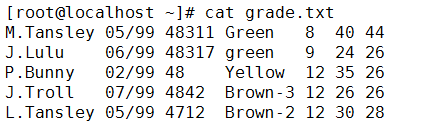
**M.Tansley 05/99 48311 Green 8 40 44**

**J.Lulu 06/99 48317 green 9 24 26**

**P.Bunny 02/99 48 Yellow 12 35 26**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**L.Tansley 05/99 4712 Brown-2 12 30 28**



此文本文件有7个域，即

（1）名字

（2）升段日期

（3）学生序号

（4）腰带级别

（5）年龄

（6）目前比赛积分

（7）比赛最高分

因为域间使用空格作为域分隔符，故不必用-F选项划分域

从windows中导入时，如果文本中有空行或特殊字符，先处理好，以方便后续操作

[root@localhost ~]# cat grade.txt |col -b >grade1.txt

[root@localhost ~]# grep -v '^$' grade1.txt >grade.txt

[root@localhost ~]# cat grade.txt

M.Tansley 05/99 48311 Green 8 40 44

J.Lulu 06/99 48317 green 9 24 26

P.Bunny 02/99 48 Yellow 12 35 26

J.Troll 07/99 4842 Brown-3 12 26 26

L.Tansley 05/99 4712 Brown-2 12 30 28

## 练习案例：

**打印出全文内容（等同于cat）：**

**[root@localhost ~]# awk '{print}' grade.txt**

**[root@localhost ~]# awk '{print}' <grade.txt**

**[root@localhost ~]# cat grade.txt |awk '{print}'**

**[root@localhost ~]# awk '{print $0}' grade.txt > wow**

**输出文件第一列和第四列：默认用逗号分隔，也可以指定分隔的字符或制表符号[root@localhost]# awk '{print $1,$4}' grade.txt**

**[root@localhost ~]# awk '{print $1"\t"$4}' grade.txt**

**M.Tansley Green**

**J.Lulu green**

**P.Bunny Yellow**

**J.Troll Brown-3**

**L.Tansley Brown-2**

**\t：代表tab键，水平制表位**

格式化输出

**[root@localhost ~]# awk 'BEGIN {print "date age\n--------"}{print $2,$5}' grade.txt**

**date age**

**--------**

**05/99 8**

**06/99 9**

**02/99 12**

**07/99 12**

**05/99 12**

**\n:代表换行**

**[root@localhost ~]# awk 'BEGIN {print "date age\n--------"}{print $2,$5}END{print "BY:Benet"}' grade.txt**

**date age**

**--------**

**05/99 8**

**06/99 9**

**02/99 12**

**07/99 12**

**05/99 12**

**BY:Benet**

**[root@localhost ~]# awk 'BEGIN {print "date age\n--------"}/Lu/{print $2,$5}END{print "BY:Benet"}'**

**date age**

**--------**

**06/99 9**

**BY:Benet**

**在输出到文件的同时输出到屏幕**

**[root@localhost ~]# awk '{print $0}' grade.txt |tee wow**

## awk条件操作符：

|  |  |
| --- | --- |
| 操作符 | 描述 |
| < | 小于 |
| > | 大于 |
| <= | 小于等于 |
| == | 等于 |
| != | 不等于 |
| >= | 大于等于 |
| ~ | 匹配正则表达式 |
| !~ | 不匹配正则表达式 |

**判断整行是否匹配Brown，匹配则输出整行，要匹配的内容（查找的内容）用//指定或用比较操作符**

[root@localhost ~]# awk '$0~/Brown/{print $0}' grade.txt

J.Troll 07/99 4842 Brown-3 12 26 26

L.Tansley 05/99 4712 Brown-2 12 30 28

**如果是整行匹配，$0~可以省略，当有编辑命令时，如果输出内容是整行，{print $0}也可以省略，默认即为输出整行**

[root@localhost ~]# awk '/Brown/' grade.txt

J.Troll 07/99 4842 Brown-3 12 26 26

L.Tansley 05/99 4712 Brown-2 12 30 28

**判断第三列是否等于48，匹配则输出整行（更加严格）**

**[root@localhost ~]# awk '$3=="48" {print $0}' grade.txt**

**P.Bunny 02/99 48 Yellow 12 35 26**

**判断整行是否不匹配Brown，不匹配则输出整行**

**[root@localhost ~]# awk '$0 !~ /Brown/' grade.txt**

**M.Tansley 05/99 48311 Green 8 40 44**

**J.Lulu 06/99 48317 green 9 24 26**

**P.Bunny 02/99 48 Yellow 12 35 26**

**[root@localhost ~]# awk '! /Brown/' grade.txt**

**M.Tansley 05/99 48311 Green 8 40 44**

**J.Lulu 06/99 48317 green 9 24 26**

**P.Bunny 02/99 48 Yellow 12 35 26**

**判断第四列是否等于Brown-2，不等于则输出整行**

**[root@localhost ~]# awk '$4 != "Brown-2"' grade.txt**

**M.Tansley 05/99 48311 Green 8 40 44**

**J.Lulu 06/99 48317 green 9 24 26**

**P.Bunny 02/99 48 Yellow 12 35 26**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**过滤文件中的Green或green**

**[root@localhost ~]# awk '/[Gg]reen/' grade.txt**

**M.Tansley 05/99 48311 Green 8 40 44**

**J.Lulu 06/99 48317 green 9 24 26**

**判断第一列第四个字母是a的**

**[root@localhost ~]# awk '$1~/^...a/' grade.txt**

**M.Tansley 05/99 48311 Green 8 40 44**

**L.Tansley 05/99 4712 Brown-2 12 30 28**

**判断整行匹配Yellow或者Brown**

**[root@localhost ~]# awk '$0~/(Yellow|Brown)/' grade.txt**

**P.Bunny 02/99 48 Yellow 12 35 26**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**L.Tansley 05/99 4712 Brown-2 12 30 28**

**[root@localhost ~]# awk '/(Yellow|Brown)/' grade.txt**

**P.Bunny 02/99 48 Yellow 12 35 26**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**L.Tansley 05/99 4712 Brown-2 12 30 28**

**过滤P开头的行**

**[root@localhost ~]# awk '/^P/' grade.txt**

**P.Bunny 02/99 48 Yellow 12 35 26**

**输出密码为空的用户行**

**[root@localhost ~]# awk -F: '$2=="!!"{print}' /etc/shadow**

**[root@localhost ~]# awk 'BEGIN{FS=":"};$2=="!!"{print}' /etc/shadow**

## awk的内置变量：

* FS： 指定每行文本的字段分隔符，缺省为空格或制表位
* OFS： 指定输出字段的分隔符，默认为空格
* NF： 当前处理的行的字段个数（列数）
* NR： 当前处理的行的序数（行数）
* FILENAME： AWK浏览的文件名
* $0： 当前处理的行的整行内容
* $n： 当前处理的第n个字段（第n列）

**统计行数和列数：**

**[root@localhost]# wc -l grade.txt**

**5 grade.txt**

**NR：内置变量NR代表当前处理的行号，如果放在END语句，就是处理完所有行后输出的当前行也就是最后一行行号，如果放在END前面就是每处理一行的当前行号，就会每行行号都显示**

**[root@localhost ~]# awk 'END {print NR}' grade.txt**

**5**

**[root@localhost ~]# awk '{print NR}' grade.txt**

**1**

**2**

**3**

**4**

**5**

**NF：内置变量NF代表的是每行的总列数（字段数），$NF代表最后一字段的内容**

**[root@localhost ~]# awk '{print NF}' grade.txt**

**7**

**7**

**7**

**7**

**7**

**[root@localhost ~]# awk 'END {print NF}' grade.txt**

**7**

**[root@localhost]# awk '{print NR,$0}END{print FILENAME}' grade.txt**

**1 M.Tansley 05/99 48311 Green 8 40 44**

**2 J.Lulu 06/99 48317 green 9 24 26**

**3 P.Bunny 02/99 48 Yellow 12 35 26**

**4 J.Troll 07/99 4842 Brown-3 12 26 26**

**5 L.Tansley 05/99 4712 Brown-2 12 30 28**

**grade.txt**

**[root@localhost ~]# awk '{print NR,$0}{print FILENAME}' grade.txt**

**1 M.Tansley 05/99 48311 Green 8 40 44**

**grade.txt**

**2 J.Lulu 06/99 48317 green 9 24 26**

**grade.txt**

**3 P.Bunny 02/99 48 Yellow 12 35 26**

**grade.txt**

**4 J.Troll 07/99 4842 Brown-3 12 26 26**

**grade.txt**

**5 L.Tansley 05/99 4712 Brown-2 12 30 28**

**grade.txt**

**[root@localhost]# awk '{print NF,NR,$0}END{print FILENAME}' grade.txt**

**7 1 M.Tansley 05/99 48311 Green 8 40 44**

**7 2 J.Lulu 06/99 48317 green 9 24 26**

**7 3 P.Bunny 02/99 48 Yellow 12 35 26**

**7 4 J.Troll 07/99 4842 Brown-3 12 26 26**

**7 5 L.Tansley 05/99 4712 Brown-2 12 30 28**

**grade.txt**

FS：内置变量FS代表分隔文件的分隔符，在命令选项中可以用-F指定，默认为空格

OFS：内置变量OFS代表输出时各个字段间的分隔符，默认为空格

**[root@localhost ~]# head -5 /etc/passwd | awk -F: '{print $1,$7}'**

**root /bin/bash**

**bin /sbin/nologin**

**daemon /sbin/nologin**

**adm /sbin/nologin**

**lp /sbin/nologin**

**[root@localhost ~]# head -5 /etc/passwd | awk 'BEGIN{FS=":"}{ print $1,$NF}'**

**root /bin/bash**

**bin /sbin/nologin**

**daemon /sbin/nologin**

**adm /sbin/nologin**

**lp /sbin/nologin**

**[root@localhost ~]# head -5 /etc/passwd | awk 'BEGIN{FS=":";OFS="^^"}{ print $1,$7}'**

**root^^/bin/bash**

**bin^^/sbin/nologin**

**daemon^^/sbin/nologin**

**adm^^/sbin/nologin**

**lp^^/sbin/nologin**

**[root@localhost ~]# cd /usr/local/etc/**

**[root@localhost etc]# echo $PWD**

**/usr/local/etc**

**[root@localhost etc]# echo $PWD | awk -F/ '{print $NF}'**

**etc**

**[root@localhost etc]# echo $PWD | awk -F/ '{print NF,$NF}'**

**4 etc**

**输出第1行~第3行的内容：**

**[root@localhost ~]# awk 'NR==1,NR==3{print}' grade.txt**

**[root@localhost ~]# awk '(NR>=1)&&(NR<=3){print}' grade.txt**

**输出第1行和第3行的内容：**

**[root@localhost ~]# awk 'NR==1||NR==3{print}' grade.txt**

**输出所有奇数行：**

**[root@localhost ~]# awk '(NR%2)==1' grade.txt**

**输出所有偶数行：**

**[root@localhost ~]# awk '(NR%2)==0' grade.txt**

**统计当前所有系统用户的用户名、UID、GID、登录的shell，制成Windows系统中的EXCEL表格：（工作中常用到）**

$7可以用$NF代替（最后一列）

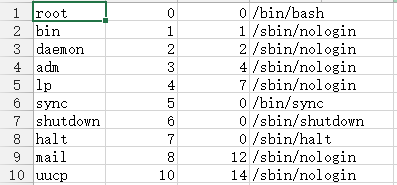
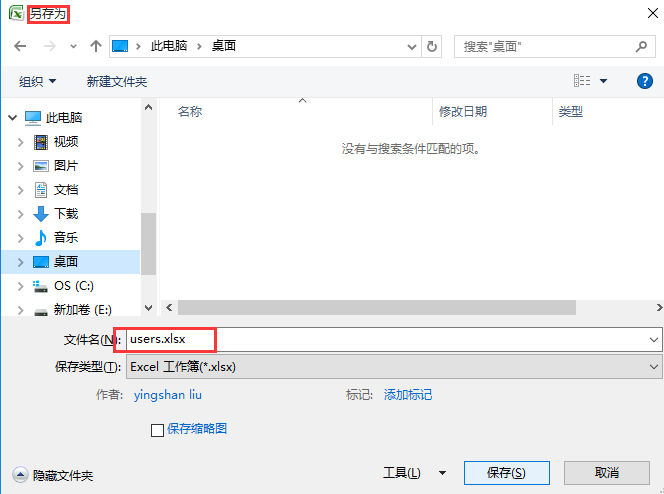
**[root@localhost ~]# awk -F: '{print $1","$3","$4","$7}' /etc/passwd >users.csv**

**[root@localhost ~]# sz users.csv**

将此文件复制到Windows电脑中



打开后，可以根据需求，修改表格，另存为成excel表格即可



**当前内存使用率超过85%时报警（取整数部分）：**

**[root@localhost ~]# free -m**

**total used free shared buff/cache available**

**Mem: 974 205 342 3 427 565**

**Swap: 2047 16 2031**

**[root@localhost ~]# vim mem.sh**

**#!/bin/bash**

**total=$(free -m | awk 'NR==2 {print $2}')**

**use=$(free -m | awk 'NR==2 {print $3}')**

**used=$(expr $use \\* 100 / $total)**

**#为了测试效果将比较值设置为10%**

**if [ $used -gt 10 ]**

**then**

**echo "内存已达到${used}%"**

**fi**

**[root@localhost ~]# bash mem.sh**

**内存已达到20%**

**[root@localhost ~]# [ $(free -m|awk '/Mem/ {print int($3/$2\*100)}') -gt 85 ] &&echo "内存使用率已超过85%！"**

## awk中引用shell变量

用单引号引用shell变量，值不能包含空格

**[root@localhost ~]# name=san**

**[root@localhost ~]# awk 'BEGIN{print "'$name'"}'**

**san**

**如果变量中有空格，需要用双引号引起来**

**[root@localhost ~]# name="zhang san"**

**[root@localhost ~]# awk 'BEGIN{print "'$name'"}'**

**awk: BEGIN{print "zhang**

**awk: ^ unterminated string**

**[root@localhost ~]# awk 'BEGIN{print "'"$name"'"}'**

**zhang san**

**[root@localhost ~]# export name="zhang san"**

**[root@localhost ~]# awk 'BEGIN{print ENVIRON["name"]}'**

**zhang san**

## awk条件判断语句

awk内置的条件语句是从C语言中借鉴过来的，可进行控制程序的流程。

单分支格式：

‘{if (表达式){

命令1;命令2; ...

}

}’

条件语句也是作为命令的一部分需要放在{}内，如果作为查找条件需要放在{}外

**判断第四列是否匹配Brown，匹配则输出整行，匹配后如果只有一条命令{}可以省略，但命令不可以省略**

**[root@localhost ~]# awk '{if($4~/Brown/) print $0}' grade.txt**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**L.Tansley 05/99 4712 Brown-2 12 30 28**

**[root@localhost ~]# awk '{if($4~/Brown/) {print $0}}' grade.txt**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**L.Tansley 05/99 4712 Brown-2 12 30 28**

**[root@server01 ~]# awk '{if($4~/Brown/) {print $0;print $1}}' grade.txt**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**J.Troll**

**L.Tansley 05/99 4712 Brown-2 12 30 28**

**L.Tansley**

**[root@server01 ~]# awk '{if($4~/Brown/) print $0;print $1}' grade.txt**

**M.Tansley**

**J.Lulu**

**P.Bunny**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**J.Troll**

**L.Tansley 05/99 4712 Brown-2 12 30 28**

**L.Tansley**

**[root@aaa ~]# awk '$3~/48/{print $0}' grade.txt**

**M.Tansley 05/99 48311 Green 8 40 44**

**J.Lulu 06/99 48317 green 9 24 26**

**P.Bunny 02/99 48 Yellow 12 35 26**

**J.Troll 07/99 4842 Brown-3 12 26 26**

如果第6个域小于第7个域则打印

**[root@localhost ~]# awk '{if ($6<$7) print $7 " too high"}' grade.txt**

**44 too hig**

**26 too high**

**判断第三列是否匹配48，匹配则输出整行**

**[root@localhost ~]# awk '{if($3~/48/) print $0}' grade.txt**

**M.Tansley 05/99 48311 Green 8 40 44**

**J.Lulu 06/99 48317 green 9 24 26**

**P.Bunny 02/99 48 Yellow 12 35 26**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**判断第四列是否匹配Brown，不匹配则输出整行**

**[root@localhost ~]# awk '{if($4!~/Brown/) print $0}' grade.txt**

**M.Tansley 05/99 48311 Green 8 40 44**

**J.Lulu 06/99 48317 green 9 24 26**

**P.Bunny 02/99 48 Yellow 12 35 26**

**判断第六列是否小于第七列，成立则输出后面信息**

**[root@localhost ~]# awk '{if ($6 < $7) print $1"\t""还有升段机会"}' grade.txt**

**M.Tansley 还有升段机会**

**J.Lulu 还有升段机会**

**判断第六列是小于等于第七列，成立则输出记录第一列**

**[root@localhost ~]# awk '{if ($6 <= $7) print $1}' grade.txt**

**M.Tansley**

**J.Lulu**

**J.Troll**

**判断第六列是大于第七列，成立则输出记录第一列**

**[root@localhost ~]# awk '{if ($6 > $7) print $1}' grade.txt**

**P.Bunny**

**L.Tansley**

**判断第一列是P.Bunny，并且第四列是Yellow的，&&是与的关系，两项要同时满足**

**[root@localhost ~]# awk '{if ($1=="P.Bunny" && $4=="Yellow") print $0}' grade.txt**

**P.Bunny 02/99 48 Yellow 12 35 26**

**判断第一列是P.Bunny，或者第四列匹配Brown的，||是或的关系，满足任意一项即可**

**[root@localhost ~]# awk '{if ($1=="P.Bunny" || $4~/Brown/) print $0}' grade.txt**

**P.Bunny 02/99 48 Yellow 12 35 26**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**L.Tansley 05/99 4712 Brown-2 12 30 28**

**[root@localhost]# awk '{if($6 < 27) print $0}' grade.txt**

**J.Lulu 06/99 48317 green 9 24 26**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**[root@localhost]# awk 'BEGIN {b="27"}{if($6 < b) print $0}' grade.txt**

**J.Lulu 06/99 48317 green 9 24 26**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**END是在前面的操作执行完成后再执行的操作，END{print $0}显示最后处理的那行内容（文本的最后一行），而不是前面符合条件的最后一行**

**[root@localhost ~]# awk 'BEGIN {basefile="27"}{if($6 < basefile) print $0}END{print $0}' grade.txt**

**J.Lulu 06/99 48317 green 9 24 26**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**L.Tansley 05/99 4712 Brown-2 12 30 28**

匹配条件的行将$6的值减1，再输出，不匹配的直接输出不减1

**[root@localhost]# awk '{if($1=="M.Tansley") $6=$6-1;print $1,$6,$7}' grade.txt**

**M.Tansley 39 44**

**J.Lulu 24 26**

**P.Bunny 35 26**

**J.Troll 26 26**

**L.Tansley 30 28**

**[root@localhost ~]# awk '{if($1=="M.Tansley") {$6=$6-1;print $1,$6,$7}}' grade.txt**

**M.Tansley 39 44**

**[root@localhost]# awk '{if($1=="J.Troll") print $1}' grade.txt**

**J.L.Troll**

**[root@localhost]# awk 'BEGIN{print "Name\t Difference"}{if($6 < $7){$8=$7--$6; print $1,$8}}' grade.txt**

**Name Difference**

**M.Tansley 4440**

**J.Lulu 2624**

**注意$7--$6是两横杠，代表将两个字段值串连起来，不是一个横杠，一横杠是减法**

**[root@localhost ~]# awk 'BEGIN{print "Name\t Difference"}{if($6 < $7){$8=$7-$6; print $1,$8}}' grade.txt**

**Name Difference**

**M.Tansley 4**

**J.Lulu 2**

**[root@localhost]# awk 'BEGIN{print "Name\t Difference\n----------------------"}{if($6 < $7){$8=$7--$6; print $1,$8}}' grade.txt**

**Name Difference**

**----------------------**

**M.Tansley 4440**

**J.Lulu 2624**

**[root@localhost]# awk 'BEGIN{print "Name\t Difference"}{if($6 < $7){diff=$7--$6; print $1,diff}}' grade.txt**

**Name Difference**

**M.Tansley 4440**

**J.Lulu 2624**

**分号代表一条命令完成，没有{}就只是作为一个设置或者条件而不是执行操作的命令，无命令时使用默认命令{print $0}**

**[root@localhost]# awk '(tot+=$6);END{print "Club student total points :" tot}' grade.txt**

**M.Tansley 05/99 48311 Green 8 40 44**

**J.Lulu 06/99 48317 green 9 24 26**

**P.Bunny 02/99 48 Yellow 12 35 26**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**L.Tansley 05/99 4712 Brown-2 12 30 28**

**Club student total points :155**

**加了{}，就是作为命令执行，大括号内多条命令用；分隔**

**[root@localhost]# awk '{(tot+=$6)};END{print "Club student total points :" tot}' grade.txt**

**Club student total points :155**

**[root@aaa ~]# awk '{(tot+=$6);print $0};END{print "Club student total points :" tot}' grade.txt**

**M.Tansley 05/99 48311 Green 8 40 44**

**J.Lulu 06/99 48317 green 9 24 26**

**P.Bunny 02/99 48 Yellow 12 35 26**

**J.Troll 07/99 4842 Brown-3 12 26 26**

**L.Tansley 05/99 4712 Brown-2 12 30 28**

**Club student total points :155**

**[root@Benet etc]# cd /boot/**

**[root@Benet boot]# ls -l |awk '/^[^d]/{print $9"\t"$5} {tot+=$5} END {print "total KB:" tot}'**

**config-2.6.18-194.el5 69593**

**initrd-2.6.18-194.el5.img 2674852**

**symvers-2.6.18-194.el5.gz 110979**

**System.map-2.6.18-194.el5 967675**

**vmlinuz-2.6.18-194.el5 1875796**

**total KB:5712207**

练习案例：

1. 计算/etc/passwd文件UID的和。

[root@localhost boot]# awk -F: '{(tot+=$3)}END{print tot}' /etc/passwd

80466

1. 计算/etc/passwd文件UID的平均值。

[root@localhost boot]#awk -F: 'BEGIN{i=0}{(num+=$3);i+=1};END{print num,i,num/i}' /etc/passwd

80466 46 1749.26

if/else语句，用于双重判断。

双分支格式：

‘{**if (表达式){  
命令1; 命令2; ...**

**}**

**else{**

**命令1; 命令2; ...**

**}**

}’

如果$7大于30则打印$1 big,否则打印$1 small。

**[root@localhost ~]# awk '{if($7>30) print $7,$1 " big";else print $7,$1 " small"}' grade.txt**

**44 M.Tansley big**

**26 J.Lulu small**

**26 P.Bunny small**

**26 J.Troll small**

**28 L.Tansley small**

if/else else if语句，用于多重判断。

多分支格式：

‘{if (表达式){

命令1; 命令2; ...

}

else if (表达式){

命令1; 命令2; ...

}

else {

命令1; 命令2; ...

}

}’

**[root@localhost ~]# awk '{if($7==44) print $1 " huge";else if($7==28) print $1 " big" ; else print $1 " small"}' grade.txt**

**M.Tansley huge**

**J.Lulu small**

**P.Bunny small**

**J.Troll small**

**L.Tansley big**

## 循环：

awk有三种循环:while循环；for循环；do循环。以while循环简单介绍

**while语句**

**格式：**

while(表达式)

{语句}

变量的初始值为1，若i小于或等于NF(记录中域的个数),则执行打印语句，且i增加1。直到i的值大于NF.

**[root@localhost ~]#** **awk 'NR==1{i=1; while(i<=NF) {print NF,$i;i++}}' grade.txt**

**7 M.Tansley**

**7 05/99**

**7 48311**

**7 Green**

**7 8**

**7 40**

**7 44**

**实现了将横向数据纵向排列**

**思考？：如果不小心写成了awk 'NR==1;{i=1; while(i<=NF) {print NR,$i;i++}}' grade.txt 多了一个分号，会有什么结果？思考原因**

**[root@localhost ~]# awk 'BEGIN{total=0; while(i<=100) {total+=i;i++;} print total,i}'**

**5050 101**

## 知识点补充：

**查看当前shell最近使用最多的10个命令：**

* sort： 排序。sort -r 反向排序；sort -n 以数字的值的大小为排序依据
* uniq： 去除重复项。uniq -c在每行前加上表示相应行目出现次数的前缀编号

**[root@localhost ~]# history |awk '{print $2}'|sort|uniq -c|sort -nr|head -10**

**[root@localhost ~]# history |awk '{print $2}'|sort|uniq -c|sort -nr|sed -n '1,10p'**

**[root@localhost ~]# history |awk '{print $2}'|sort|uniq -c|sort -nr|awk 'NR<=10{print}'**

**[root@localhost ~]# history |awk '{print $2}'|sort|uniq -c|sort -nr|awk 'NR==1,NR==10{print}'**

**43 awk**

**17 sed**

**12 grep**

**7 cat**

**5 ifconfig**

**4 vim**

**3 nl**

**2 ip**

**2 free**

**1 yum**

**将文本中纵向数据横向显示，默认sed每处理一行就会自动添加一个行的分隔符即换行符，如果要合并行只需要使sed处理合并多行后再替换换行符即可，N;代表先把下一行合并为一行再处理，多个N;可以合并多行，然后将这个合并行里的换行符替换为空，输出时就没有换行符了**

[root@localhost ~]# cat test

1

2

a

b

x

y

[root@CentOS6-node1 ~]# sed 'N;s/\n/ /g' test

1 2

a b

x y

[root@localhost ~]# sed 'N;N;s/\n/ /g' test

1 2 a

b x y

**xargs:（英文全拼： eXtended ARGuments）是给命令传递参数的一个过滤器，也是组合多个命令的一个工具。**

xargs 可以将管道或标准输入（stdin）数据转换成命令行参数，也能够从文件的输出中读取数据。

xargs 也可以将单行或多行文本输入转换为其他格式，例如多行变单行，单行变多行。

xargs 默认的命令是 echo，这意味着通过管道传递给 xargs 的输入将会包含换行和空白，不过通过 xargs 的处理，换行和空白将被空格取代，因此默认是一行输出所有内容，可以用-n选项指定需要在一行输出的原来行数内容。

xargs 是一个强有力的命令，它能够捕获一个命令的输出，然后传递给另外一个命令。

之所以能用到这个命令，关键是由于很多命令不支持|管道来传递参数，而日常工作中有有这个必要，所以就有了 xargs 命令，例如：

find /etc -name 'yum.\*'| ls -l #这个命令是错误的

find /etc -name 'yum.\*'| xargs ls -l #这样才是正确的

xargs 一般是和管道一起使用。

[root@localhost ~]# cat test |xargs

1 2 a b x y

[root@localhost ~]# cat test |xargs -n 2

1 2

a b

x y